DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico III

System Programming - Zombi defense

Organización del Computador II

Segundo Cuatrimestre de 2014

Integrante	LU	Correo electrónico
Aldasoro Agustina	86/13	agusaldasoro@gmail.com
Rey Maximiliano	XXX/XX	mail
Tirabasso Ignacio	XXX/XX	mail

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

1. Ejercio 1

a) Armamos los cuatro segmentos de la GDT, llamándolos:

```
    [GDT_IDX_CODE_0] = (gdt_entry) ;
    [GDT_IDX_CODE_3] = (gdt_entry) ;
    [GDT_IDX_DATA_0] = (gdt_entry) ;
    [GDT_IDX_DATA_3] = (gdt_entry) ;
```

A los cuatro les seteamos el mismo *límite*: 0x26EFF = ... , y la misma *base* en 0. El *segment type* varía depende el segmento: CODE_0: 0x0A= ... , CODE_3: 0x0F= ... , DATA_0 y DATA_3: 0x02= ... . El *Descriptor type* va en todos para system, por lo tanto es 0. El *Descriptor privilege level* coincide con el nombre del descriptor (0 para CODE_0 y DATA_0; 3 para CODE_3 y DATA_3). El bit de *Present* va para todos en 1 y los bit de *Available for use by system software* y *l* van para todos en 0. El bit de *Default operation size* va para todos en 1 porque es un código de 32bits. El bit de *Granularity* va para todos en 1.

b) Se adjunta el código necesario para pasar a modo protegido y setear la pila del kernel en la dirección 0x27000.

```
    ; Deshabilitar interrupciones
    cli

    ; Habilitar A20
    call habilitar_A20

    ; Cargar la GDT
    lgdt [GDT_DESC]

    ; Setear el bit PE del registro CR0
    mov eax,cr0
    or eax,1
    mov cr0,eax

    jmp 0x50:modo_protegido
```

```
BITS 32
modo_protegido:

    ; Establecer selectores de segmentos
    xor eax, eax
    mov ax, 0x40

    mov es, ax
    mov ds, ax
    mov ss, ax
    mov gs, ax

    mov ax, 0x60
    mov fs, ax

    ; Establecer la base de la pila
    mov ebp, 0x27000
```

c) Segmento adicional que describe el área de la pantalla en memoria que puede ser utilizado solo por el kernel:

```
    ; Cambiar modo de video a 80 X 50
    mov ax, 0003h
    int 10h ; set mode 03h
    xor bx, bx
    mov ax, 1112h
    int 10h ; load 8x8 font
```

d) La siguiente es la rutina que se encarga de limpiar la pantalla y pintar el área del mapa un fondo de color verde, junto con las dos barras laterales para cada uno de los jugadores (rojo y azul).

```
    ; Inicializar pantalla
    call clear_screen
    call print_map ;código que va a pintar el mapa de los colores deseados
```

```
void clear_screen() {
    int size = VIDEO_COLS * VIDEO_FILS;
    ca (*p) = (ca (*)) VIDEO;
    int i = 0;
    ca empty;
    empty.c = 0;
    empty.a = getFormat(C_FG_BLACK, 0, C_BG_BLACK, 0);
    while(i < size) {
        p[i] = empty;
        i++;
    }
}
```

2. Ejercio 2

a) Inicializamos las entradas 0-19 en la IDT.
b) Para probar lo programado anteriormente hicimos:

```
; Inicializar la IDT
call idt_inicializar
; Cargar IDT
lidt [IDT_DESC]

; test para que salte la divide by 0 exception (0)
mov edx,0
mov ecx,0
mov eax,3
div ecx
```

3. Ejercio 3

a) La siguiente es la rutina que se encarga de limpiar el buffer de video y pintarlo como indica la figura 9.

```
void print_map() {
    int cols = VIDEO_COLS;
    int rows = VIDEO_FILS;

    ca (*screen)[VIDEO_COLS] = (ca (*)[VIDEO_COLS]) VIDEO;

/* Defino los colores que voy a usar */
    ca red;
    red.c = 0;
    red.a = getFormat(C_FG_RED, 0, C_BG_RED, 0);
    ca blue;
    blue.c = 0;
    blue.a = getFormat(C_FG_BLUE, 0, C_BG_BLUE, 0);
    ca green;
    green.c = 0;
    green.a = getFormat(C_FG_GREEN, 0, C_BG_GREEN, 0);
    ca black;
    black.c = 0;
    black.a = getFormat(C_FG_BLACK, 0, C_BG_BLACK, 0);

    int y,x;

    clear_screen();

    for(y = 0; y < rows; y++) {
        for(x = 0; x < cols; x++) {
            if (y >= rows-5) {
                screen[y][x] = black; //las últimas 5 filas van de negro
            } else if (x == cols-1) {
                screen[y][x] = blue; //el jugador de la derecha es azul
            } else if (x == 0) {
                screen[y][x] = red; //el jugador de la izquierda es rojo
            } else {
                screen[y][x] = green; //el terreno de juego es verde
            }
        }
    }
    /*Armo el cuadrado para el puntaje rojo */
    for(y = rows-5; y < rows; y++) {
        for(x = 35; x < 40; x++) {
            screen[y][x] = red;
        }
    }
    /*Armo el cuadrado para el puntaje azul */
    for(y = rows-5; y < rows; y++) {
        for(x = 40; x < 45; x++) {
            screen[y][x] = blue;
        }
    }
}
```

b) Se muestran las rutinas encargadas de inicializar el directorio y tablas de páginas para el kernel (mmu_inicializar_dir_kernel).

```
void mmu_inicializar_dir_kernel() {
    page_directory *pd = (page_directory *) 0x27000;
    int i;

    /* Creo 1024 entradas en page_directory con todo cero. */
    for (i = 0; i < 1024; i++) {
        pd[i] = (page_directory) {};
    }

    /* Mapeo las primeras 4 (0,1,2,3) entradas del page directory
    con la base y permisos correspondientes. */
    for(i = 0; i < 4; i++) {
        pd[i] = (page_directory) {
            .base = 0x28 + i,
            .rw = 0x1,
            .p = 1,
        };
    }

    page_table* pt = (page_table*) 0x28000;
    for(i = 0; i < 1024; i++) {
        pt[i] = (page_table) {
            .base = i,
            .rw = 0x1,
            .p = 1,
        };
    }
}
```

c) Completar el código necesario para activar paginación.

```
    ; Habilitar paginacion

    mov eax,0x27000
    mov cr3,eax

    mov eax,cr0
    or eax,0x80000000
    mov cr0,eax


    mov eax,0x100000
    mov cr3,eax
```

4. Ejercio 4

a) Para administrar la memoria en el área libre, tenemos un contador de páginas utilizadas denominándolo *páginas*. Luego contamos con las funciones *get_page_directory* y *get_page_table* las cuales nos brindan un nuevo page directory o una nueva page table correspondientemente.

```
page_directory* get_page_directory() {
    page_directory* pd = (page_directory*) PAGES;
    pd += paginas * 4096;
    int i;
    for (i = 0; i < 1024; i++) {
        pd[i] = (page_directory) {};
    }

    pd[0] = (page_directory) {
        .base = 0x28,
        .rw = 0x1,
        .p = 1
    };

    paginas++;

    return pd;
}
```

```
page_table* get_page_table() {
    page_table* pgt = (page_table*) PAGES;
    pgt += paginas * 0x1000;
    int i;
    for (i = 0; i < 1024; i++) {
        pgt[i] = (page_table) {};
    }

    paginas++;

    return pgt
    };
}
```

b) La rutina *mmu_inicializar_dir_zombi* se encarga de inicializar un directorio de páginas y tablas de páginas para una tarea, respetando la figura 6. Copia el código de la tarea a su área asignada, es decir la posición indicada por el jugador dentro del mapa y mapea dichas páginas a partir de la dirección virtual 0x08000000(128MB).

```
/* Guerrero = 0, Mago = 1, Clerigo = 2
   player = 0 es A
   player = 1 es B
*/
  page_directory* mmu_inicializar_dir_zombie(unsigned int player, unsigned char class,
  unsigned int y) {

    page_directory* pd = get_page_directory();
    unsigned int x = (player ? 79 : 2);

    unsigned int offset_x[9] = {0, -1, -1, -1, 0, 0, 1, 1, 1};
    unsigned int offset_y[9] = {0, 0, -1, 1, -1, 1, 0, 1, -1};

    /* En este for se mapean las nueve páginas correspondientes a un jugador */
    int i, _x, _y;
    for(i = 0; i < 9; i++) {
        _x = y + offset_x[i] * (player ? 1 : -1);  \\ x es la posición x dentro del mapa
        _y = x + offset_y[i] * (player ? 1 : -1);  \\ y es la posición y dentro del mapa

        /* La función get_physical_address devuelve la dirección física en memoria
        a la cual le corresponde el par (x, y) pasado por parámetro */
        mmu_mapear_pagina(0x8000000 + (i*0x1000), pd, get_physical_address(_x, _y), 1, 0);
    }

    /* Código para copiar la tarea del Zombie */
    int address = address = 0x10000 + (player ? 0 : 1)  * 0x3000 + class * 0x1000;
    i = 0;
    unsigned char *code = (unsigned char *) 0x8000000;
    unsigned char * paddress = (unsigned char*) address;
    while (i++ < 0x1000) {
        code[i] = paddress[i];
    }

    return pd;
}
```

c) La rutina *mmu_mapear_página* permite mapear la página física correspondiente a su física en la dirección virtual utilizando cr3.

```
void mmu_mapear_pagina(unsigned int virtual, page_directory* pd, unsigned int fisica,
   unsigned char rw, unsigned char us) {

    unsigned int directory = (virtual >> 22);
    unsigned int table     = (virtual & 0x003FF000) >> 12;

    page_table* pt = (page_table*) (pd[directory].base << 12);

    if (pd[directory].p == 0){
        pd[directory].base = ((unsigned int) get_page_table()) >> 12;
        pd[directory].rw = rw;
    pd[directory].us = us;
        pd[directory].p = 1;
    }

    pt = (page_table*) (pd[directory].base << 12);
    pt[table].base = fisica >> 12;
    pt[table].rw = rw;
    pt[table].us = us;
    pt[table].p = 1;

    tlbflush();
}
```

Esto no lo hicimos II- mmu unmapear pagina(unsigned int virtual, unsigned int cr3) Borra el mapeo creado en la dirección virtual virtual utilizando cr3.

```
Aca va la implementacion de muu_unmapear_pagina
```

5. Ejercio 5

a) Completamos las entradas necesarias en la IDT para asociar una rutina a la interrupción del reloj, otra a la interrupción de teclado y por último una a la interrupción de software 0x66. Es decir las posiciones 32,33 y 66.

b) A continuación, la rutina asociada a la interrupción del reloj, para que por cada tick llame a la función screen próximo reloj. La misma se encarga de mostrar cada vez que se llame, la animación de un cursor rotando en la esquina inferior derecha de la pantalla.

```
;; Rutina de atención del RELOJ

global _isr32
_isr32:
    pushad
    call proximo_reloj   ; Ya definida en isr.asm
    call proximo_indice  ;Devuelve el próximo índice en la GDT a ejecutar

    cmp ax,0
    je .nojump

    mov [sched_tarea_selector], ax
    call fin_intr_pic1
    jmp far [sched_tarea_offset]
    jmp .end

.nojump:
    call fin_intr_pic1

.end:
    ; switchear tareas.
    popad
    iret
```

c) Ahora, la rutina asociada a la interrupción de teclado de forma que si se presiona cualquiera de las teclas a utilizar en el juego, se presenta la misma en la esquina superior derecha de la pantalla.

```
;; Rutina de atención del TECLADO

global _isr33
extern printf
extern print_int
extern handle_keyboard_interrumption
_isr33:
    pushad
    xor eax,eax
    in al, 0x60

    mov dword [esp], eax
    call handle_keyboard_interrumption

    mov dword  [esp + 0x], 0
    mov dword [esp + 0xc], 67
    mov dword [esp + 0x8], keyboard_str
    mov dword [esp + 0x4], eax
    call printf    ;función implementada por nosotros

    call fin_intr_pic1
    popad
    iret
```

d) Escribir la rutina asociada a la interrupción 0x66 para que modifique el valor de eax por 0x42. Posteriormente este comportamiento va a ser modificado para atender el servicio del sistema.

Esto no lo hicimos tampoco!

6. Ejercio 6

a) Definimos tres entradas en la GDT que consideramos necesarias para ser usadas como descriptores de TSS: una para ser utilizada por la tarea inicial, otra para la tarea actual y una última para la tarea siguiente.

b) Completamos la entrada de la TSS de la tarea Idle con la información de la tarea Idle. La tarea Idle se encuentra en la dirección 0x00016000. La pila se alojará en la misma dirección que la pila del kernel y será mapeada con identity mapping. Esta tarea ocupa 1 pagina de 4KB y debe ser mapeada con identity mapping. Además la misma comparte el mismo CR3 que el kernel.

```
void tss_inicializar() {
    int i = 0;
    while(i < CANT_ZOMBIS) {
        inUseA[i] = 0;
        inUseB[i] = 0;
        i++;
    }
    currentZombieA = 0;
    currentZombieB = 0;

    // inicializar tss_idle
    tss_inicializar_tarea_idle();

    memcpy(&tss_idle, &tss_inicial, sizeof(tss));
    memcpy(&tss_idle, &current_task, sizeof(tss));
    memcpy(&tss_idle, &next_task, sizeof(tss));

    gdt[GDT_INITIAL_TSS].base_31_24 = ((u32) (&tss_inicial) & 0xFF000000) >> 24;
    gdt[GDT_INITIAL_TSS].base_23_16 = ((u32) (&tss_inicial) & 0x00FF0000) >> 16;
    gdt[GDT_INITIAL_TSS].base_0_15  = (u32) (&tss_inicial) & 0x0000FFFF;

    gdt[GDT_CURRENT_TSS].base_31_24 = ((u32) (&current_task) & 0xFF000000) >> 24;
    gdt[GDT_CURRENT_TSS].base_23_16 = ((u32) (&current_task) & 0x00FF0000) >> 16;
    gdt[GDT_CURRENT_TSS].base_0_15  = (u32) (&current_task) & 0x0000FFFF;
}
```

```
void tss_inicializar_tarea_idle() {

    tss_idle = (tss) {};

    tss_idle.eip = 0x00016000;
    tss_idle.cr3 = 0x27000;

    tss_idle.ebp = 0x27000;
    tss_idle.esp = 0x27000;

    tss_idle.es = 0x40;
    tss_idle.ds = 0x40;
    tss_idle.ss = 0x40;
    tss_idle.gs = 0x40;
    tss_idle.cs = 0x50;

    tss_idle.eflags = 0x202;
    tss_idle.iomap = 0xffff;
}
```

c) Construir una función que complete una TSS libre con los datos correspondientes a una tarea (zombi). El código de las tareas se encuentra a partir de la dirección 0x00010000 ocupando una pagina de 4kb cada una según indica la figura 1. Para la dirección de la pila se debe utilizar el mismo espacio de la tarea, la misma crecerá desde la base de la tarea. Para el mapa de memoria se debe construir uno nuevo utilizando la función mmu inicializar dir zombi. Además, tener en cuenta que cada tarea utilizará una pila distinta de nivel 0, para esto se debe pedir una nueva pagina libre a tal fin.

Y aca que onda?

d) Código necesario para ejecutar la tarea Idle, es decir, saltar intercambiando las TSS, entre la tarea inicial y la tarea Idle:

```
idle:
    .loopear:
        inc dword [numero]
        cmp dword [numero], 0x4
        jb .imprimir

    .reset_contador:
        mov dword [numero], 0x0

    .imprimir:
        ; Imprimir 'reloj'
        mov ebx, dword [numero]
        add ebx, message1
        imprimir_texto_mp ebx, 1, 0x0f, 49, 76
        mov ebx, chirimbolo_open
        imprimir_texto_mp ebx, 1, 0x0f, 49, 76-1
        mov ebx, chirimbolo_close
        imprimir_texto_mp ebx, 1, 0x0f, 49, 76+1

    jmp .loopear
```

7. Ejercio 7

a) Construir una función para inicializar las estructuras de datos del scheduler.

b) Crear la función sched próximo indice() que devuelve el índice en la GDT de la próxima tarea a ser ejecutada. Construir la rutina de forma devuelva una tarea de cada jugador por vez según se explica en la sección 3.2

c) Modificar la rutina de la interrupción 0x66, para que implemente el servicio mover según se indica en la sección 3.1.1.

d) Modificar el código necesario para que se realice el intercambio de tareas por cada ciclo de reloj. El intercambio se realizará según indique la función sched proximo índice().

e) Modificar las rutinas de excepciones del procesador para que desalojen a la tarea que estaba corriendo y corran la próxima.

f) Implementar el mecanismo de debugging explicado en la sección 3.4 que indicará en pantalla la razón del desalojo de una tarea.

8. Ejercio 8

a) Crear un conjunto de 3 tareas zombis (Guerrero, Mago y Clerigo). Los mismos deberán respetar las restricciones del trabajo práctico, ya que de no hacerlo no podrán ser ejecutados en el sistema implementado por la cátedra.

Deben cumplir:

No ocupar más de 4 kb cada uno (tener en cuenta la pila).

Tener como punto de entrada la dirección cero.

Estar compilado para correr desde la dirección 0x08000000.

Utilizar el unico servicio del sistema (mover).

Explicar en pocas palabras qué estrategia utiliza cada uno de los zombis, o en su conjunto en términos de defensa y ataque.

b) Si consideran que sus tareas pueden hacer algo mas que completar el primer item de este ejercicio, y tienen a un audaz campion que se atreva a enfrentarse en el campo de batalla zombi, entonces pueden enviar el binario de sus tareas a la lista de docentes indicando los siguientes datos:

Nombre del campion (Alumno de la materia que se presente como jugador)

Nombre de cada uno de las tareas zombi

Estrategia de alimentación de los zombis (es decir, como se comerán los cerebros de las otras tareas)

Se realizará una competencia a fin de cuatrimestre con premios en/de chocolate para los primeros puestos.

c) Pelicula y Video Juego favorito sobre Zombis.

9. Conclusiones y trabajo futuro